Título: "SmartTrack: Mobilidade Sustentável no Campus"

Apresentação do Projeto

Imagine um campus universitário movimentado onde os estudantes correm entre as aulas, apenas para encontrar ônibus lotados ou atrasados. A frustração aumenta à medida que os horários se chocam e os ônibus vazios desperdiçam combustível em rotas subutilizadas. Agora imagine uma solução: ônibus que *sabem* quando e onde são necessários, transmitindo em tempo real a ocupação, a qualidade do ar e os tempos estimados de chegada para os celulares dos estudantes. Este é o **SmartTrack**—um sistema nascido da fusão de sensores de baixo custo, IA e tecnologia sem fio, projetado para transformar o trânsito caótico do campus em uma experiência tranquila e sustentável.

Escopo do projeto

Objetivos do Projeto

1. **Rastreamento em Tempo Real e Transparência**:

- Fornecer aos estudantes localizações de ônibus ao vivo, estimativas de ocupação e horários de chegada através de um aplicativo amigável.
- Monitorar o conforto ambiental (qualidade do ar, temperatura) para melhorar o bem-estar dos passageiros.

2. **Eficiência Impulsionada por IA**:

- Prever padrões de lotação usando aprendizado de máquina para ajustar dinamicamente rotas e horários.
- Reduzir o desperdício de combustível e emissões otimizando a implantação dos ônibus.

3. **Segurança e Escalabilidade**:

- Detectar comportamentos dos passageiros (por exemplo, superlotação) via sensores para garantir a segurança.
- Criar um sistema modular adaptável a outros campi ou redes de transporte público.

Como Funciona

Camada 1: Os Sensores

- **GPS & LoRa**: Os ônibus transmitem sua localização periodiacamente via a rede de longo alcance e baixo consumo de energia LoRa.
- **Detecção de Ocupação**: Sensor de peso YZC-133 (50KG) + HX711 dentro de tapetes de borracha para estimar os passageiros embarcando/desembarcando na entrada dos ônibus.
- **Monitoramento de Conforto**: O BME680 monitora a qualidade do ar (CO_2, VOCs), e o MPU6050 detecta movimentos bruscos (por exemplo, frenagens bruscas).
- **Previsões inteligêntes**: Com os dados coletados e enviados pelo sistema embarcados será feito previsões horarios de pico nos onibus universitarios via servidor e mostrado na interface de usuário.

Camada 2: Processamento Embarcado e Nuvem

- A bordo do Raspberry Pi Pico, o sistema gerencia as tarefas:
  - Coleta de dados dos sensores (Leitura de temperatura, atualizações de GPS).
  - Processamento dos dados coletados para estimativa de pessoas e monitoramento de conforto.
  - Transmissões LoRa para enviar dados processados no micro-controlador.
- IA no Servidor: Dados históricos e em tempo real treinam redes neurais para prever picos de demanda, sugerindo ajustes de rota aos despachantes.

Camada 3: O Aplicativo
Os estudantes abrem o aplicativo SmartTrack para ver:

- Um mapa ao vivo dos ônibus codificados por cores de acordo com a ocupação (verde = assentos disponíveis, vermelho = cheio).
- Tempos estimados de chegada, classificações de qualidade do ar e alertas para atrasos ou problemas de segurança.
- Um recurso “Planeje Minha Viagem” que os alunos informam quais intercampis e quando irão utilizar e recomenda horários de partida para evitar super lotamento nos intercampis.
- Através desse recurso "Planeje Minha Viagem" os servidores vão poder fazer uma boa estimativa de pessoas que irão utilizar os intercampis e em qual horario.

Por que SmartTrack?

1. **Impacto Ambiental**:

- As metricas geradas pelos sensores do SmartTrack poderão otimizar rotas que podem reduzir as emissões de gases poluentes.

2. **Segurança dos Estudantes**:
   Após a pandemia, a superlotação continua sendo uma preocupação de saúde. Dados de ocupação em tempo real permitem que os estudantes evitem ônibus lotados.
3. **Eficiência de Custos**:
   Varias Universidade pública desperdiçam milhares de reais em rotas subutilizadas no Brasil. A IA preditiva no servidor fornece dados para que os ônibus sejam implantados onde a demanda é maior. Assim, economizando milhares de reais em dinheiro público em combustivel ao ano e milhões se for implantado em varias universidades.

Originalidade: Sobre os Ombros de Gigantes

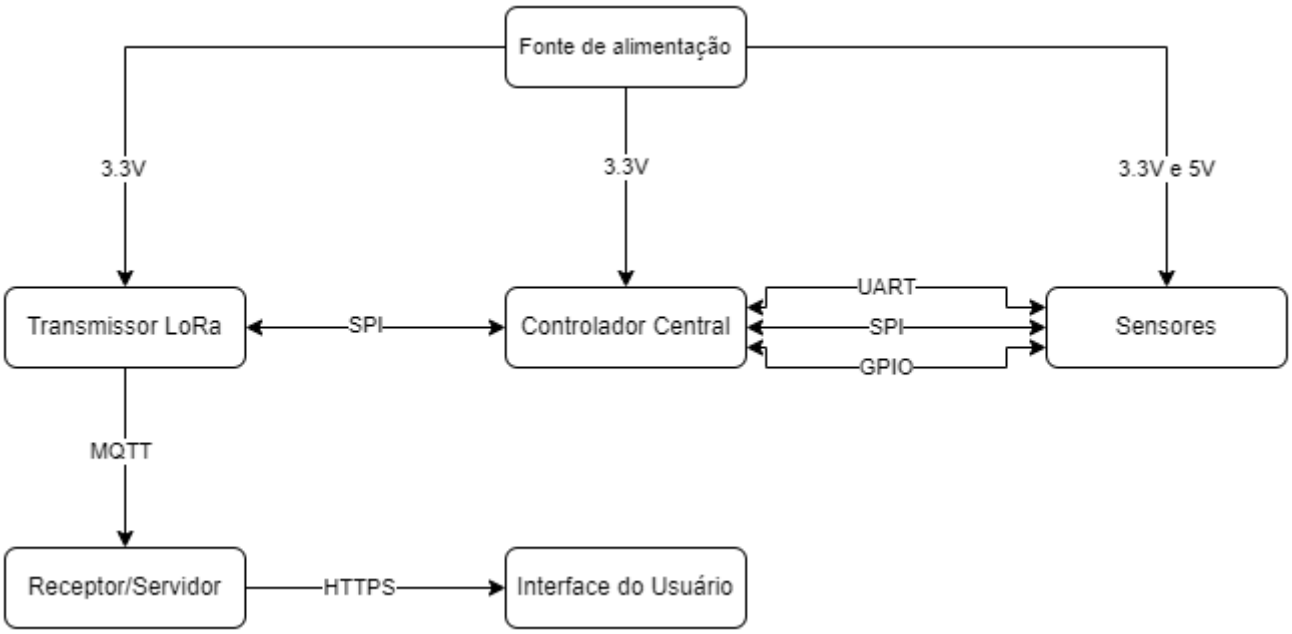
Embora projetos semelhantes existam, o SmartTrack ultrapassa limites:

- **Soluções Existentes**:
  - *TransLoc* (usado pela Universidade Duke): Rastreamento por GPS, mas sem sensores ambientais. [1]
  - *Moovit*: Dados de ocupação por crowdsourcing, sem ajustes em tempo real via sensores. [2]
  - *Rastreamento de Ônibus Baseado em IoT*: Muitos projetos acadêmicos usam GPS/LoRa, mas omitem previsões impulsionadas por IA. [3]

- **Vantagem Única do SmartTrack**:
  - **Sensoriamento Multimodal**: Combina estimativa por sensores de peso nas portas dos ônibus e previsões de uso e horário dos intercampis enviadas pelos alunos pelo aplicativo para prever com IA no servidor qual o número mais próximo de quantidade de alunos em cada intercampi.
  - **Métricas de Sustentabilidade**: Monitoramento da qualidade do ar vincula a eficiência do trânsito à saúde dos estudantes, sendo essa uma lacuna nas ferramentas existentes.

Especificações do Hardware

1. Diagrama de Blocos



Descrição:
O sistema é dividido em **7 blocos principais**, interconectados via protocolos de comunicação e pinos GPIO:

1. **Controlador Central**: Raspberry Pi Pico (microcontrolador RP2040).
2. **Sensores**: Módulo GPS, BME680 (Qualidade do ar), YZC-133 (50KG) + HX711 (Sensor de peso), MPU6050 (IMU).
3. **Transmissor LoRa**: Semtech SX1276 para comunicação de longa distância.
4. **Fonte de Alimentação**: Alimentação da energia 12V ou 5V do ônibus passando por um regulador LD1117V33 para 5V e 3.3V para os componentes embarcados.
5. **Receptor/Servidor**: Um microcontrolador com uma antena e receptor LoRa conectado a internet para repassar as mensagens LoRa MQTT dos sistemas embarcados nos intercampis para o servidor.
6. **Interface do Usuário**: Aplicativo móvel (Bloco indireto via link LoRa-para-servidor).

---

2. Função de Cada Bloco

Controlador Central:
O Raspberry Pi Pico atua como o processador central, gerenciando sensores, processamento e comunicação LoRa.

Sensores:
No bloco de sensores estão basicamente incluídos todos os componentes conectados na Raspberry Pi Pico. O módulo GPS fornece dados de latitude/longitude em tempo real usando o protocolo NMEA, enquanto o BME680 monitora a qualidade do ar, incluindo VOCs, CO_2, temperatura e umidade. O processamento dos dados gerados pelos sensores de peso nos degraus da escada do intercampi estima os passageiros que entram e saem do ônibus, e o MPU6050 detecta movimentos bruscos, como frenagens bruscas, através de seu acelerômetro e giroscópio.

Transmissor Lora:
O transmissor LoRa nos intercampis transmite os dados dos sensores para um receptor LoRa conectado a um micro controlador (por exemplo o Raspberry Pi Pico) que transmite via wifi para um servidor, permitindo acesso ao servidor ou à nuvem.

Fonte de Alimentação:
A fonte de alimentação converte a entrada de 12V ou 5V do ônibus para 3.3V e 5V, utilizando um regulador LD1117V33. Isso garante que o Raspberry Pi Pico e todos os sensores recebam a tensão adequada para operação estável. Capacitores de desacoplamento (100nF) são usados em todos os ICs para reduzir ruídos e garantir a integridade do sinal.

Interface do Usuário:
O aplicativo SmartTrack oferece uma interface intuitiva e amigável para os estudantes, exibindo a localização dos ônibus em tempo real com codificação por cores para indicar a ocupação. Além disso, os tempos estimados de chegada são mostrados nas paradas selecionadas, com notificações push alertando sobre a chegada iminente ou atrasos.

O aplicativo também exibe classificações de qualidade do ar dentro dos ônibus, incluindo níveis de CO_2 e VOCs, além de informações sobre temperatura e umidade. A funcionalidade "Planeje Minha Viagem" permite que os estudantes insiram seus horários e destinos para receber recomendações sobre os melhores horários de partida, utilizando dados históricos e em tempo real para sugerir rotas e horários otimizados.

Para segurança, o aplicativo notifica os usuários sobre problemas detectados, como superlotação ou frenagens bruscas, e inclui um botão de emergência para reportar incidentes. O histórico de viagens mantém um registro das viagens anteriores, oferecendo feedback sobre a eficiência das rotas e a qualidade do serviço. As configurações do usuário permitem personalizar preferências de notificação e visualização, incluindo temas de cores e frequência das atualizações.

---

3. Configuração de cada bloco

Bloco	Configuração
Raspberry Pi Pico	Cortex-M0+ de núcleo duplo a 125 MHz; tarefas para pesquisa, processamento e comunicação de sensores.
Modulo GPS	Interface UART rodando a 9600 de baud rate; codigos NMEA analisadas via biblioteca TinyGPS.
BME680	Interface I2C (endereço `0x76`); configurada para intervalo de amostragem periodico.

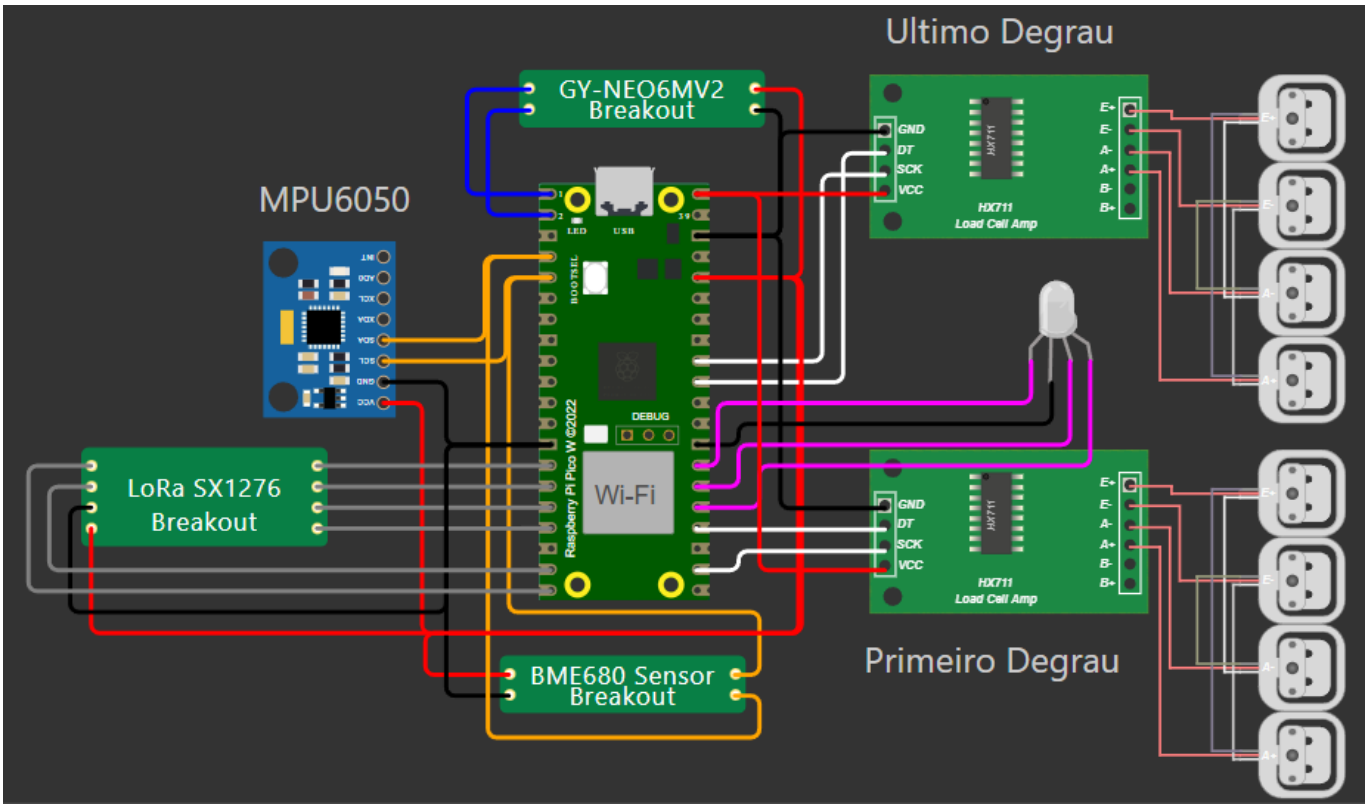
Bloco	Configuração
YZC-133 (50KG) + HX711	Interface com HX711 (ADC de 24 bits); configurado para amostras por segundo e ganho padrões.
MPU6050	Interface I2C (endereço `0x68`); alcance do acelerômetro ±2g, giroscópio ±250°/s.
Transmissor LoRa	Interface SPI (8MHz); configurada para frequência de 915MHz (EUA), SF=7, BW=125kHz.
Fonte de Alimentação	12V/5V Intercampi → LD1117V33 regulador de 3,3V; capacitores de desacoplamento (100nF) em todos os CIs e Raspberry Pi Pico.

4. Descrição do Pinout

Raspberry Pi Pico Pinout:

Pino	Função	Conectado em
GP0/GP1	UART0 (TX/RX)	Modulo GPS
GP2/GP3	I2C1 (SDA/SCL)	BME680, MPU6050
GP10-13	SPI1 (SCK/MOSI/MISO/NSS)	Transmissor LoRa
GP14-15	RESET e DIO0	Transmissor LoRa
GP19	Controle Led Azul	RGB BLUE
GP20	Controle Led Verde	RGB GREEN
GP21	Controle Led Vermelho	RGB RED
GP26	Primeiro Degrau Data (DT)	HX711 (DT)
GP27	Primeiro Degrau Clock (SCK)	HX711 (SCK)
GP18	Segundo Degrau Data (DT)	HX711 (DT)
GP27	Segundo Degrau Clock (SCK)	HX711 (SCK)
3V3	Saida 3.3V	Todos sensores
GND	Ground	Todos sensores

5. Circuito Completo de Hardware



Esquema de cores dos fios:

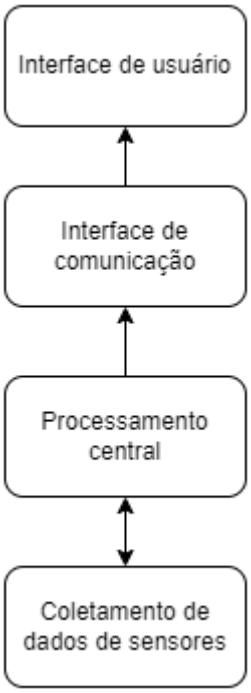
Cor	Função
Vermelho	VCC
Preto	GND
Azul	UART
Laranja	I2C
Cinza	SPI
Rosa	Controle de Leds
Branco	Sinal Dados HX711 e Clock

O ideal é adicionar um resistor pull-up de 2.7kΩ no barramento I2C e um capacitor de desacoplamento de 100uF nos pinos 3V3 e GND.
Para fins de simplificação, foi simplificado bastante a parte eletrônica para poupar tempo e tentar deixar as coisas mais organizadas. Sem considerar que o wokwi não permite certas coisas nesse sentido.

Especificações de Firmware

---

1. Blocos Funcionais



---

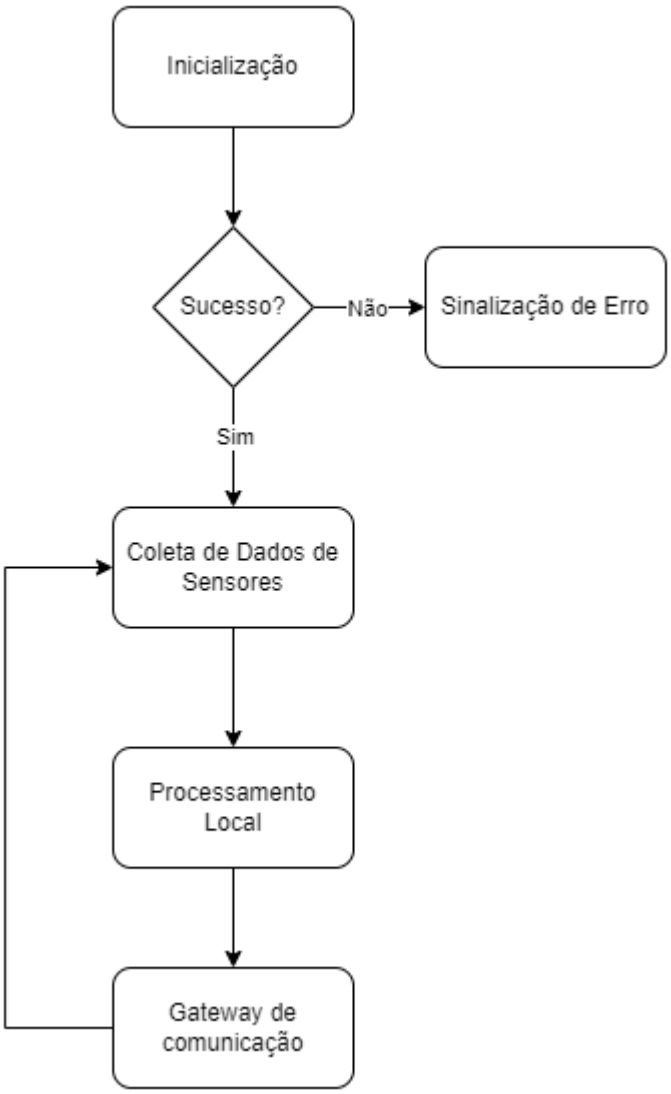
2. Descrição Funcional

- **Coletamento de dados de sensores**:
  - Monitora localização GPS (latitude/longitude).
  - Coleta dados qualidade do ar usando BME680 (CO_2, VOCs, temperatura, umidade).
  - Detecta estimativa de ocupação de passageiros usando YZC-133 (50KG) + HX711 na porta dos intercampis.
  - Monitora movimento e vibração do ônibus com sensor IMU MPU6050.
- **Central de processamento**:
  - Faz o Poll de cada sensor em intervalos.
  - Usa o módulo LoRa conectado à Raspberry Pi Pico para transmissão de dados.
  - Processa dados dos sensores na placa (Ex.: Batidas de Ônibus, Alertas de Superlotação).
- **Camada de interface de comunicação**:
  - Transmite dados de sensores para um transmissor LoRa via interface SPI.
  - A Raspberry Pi Pico processa os dados localmente e estima a quantidade de alunos no intercampi e manda os dados para a interface de usuário.
- **Camada de interface de usuário**:
  - Um mapa ao vivo dos ônibus codificados por cores de acordo com a ocupação (verde = assentos disponíveis, vermelho = cheio).
  - Tempos estimados de chegada, classificações de qualidade do ar e alertas para atrasos ou problemas de segurança.
  - Um recurso “Planeje Minha Viagem” que os alunos informam quais intercampis e quando irão utilizar e recomenda horários de partida para evitar superlotamento nos intercampis.

- Através desse recurso "Planeje Minha Viagem" os servidores vão poder fazer uma boa estimativa de pessoas que irão utilizar os intercampis e em qual horário.

---

4. Flowchart



O fluxograma descreve o fluxo de trabalho completo do software:

1. **Inicialização**:
    - Inicializa os pinos de hardware e configura módulos de comunicação.
    - Reseta todos os sensores e módulos para garantir que não reste nenhum dado residual.
2. **Sinalização de Erro**:
    - Quando qualquer inicialização de hardware não houver sucesso liga o LED vermelho e desliga quando o sistema ser reinicializado.
3. **Coleta de Dados de Sensores**:
    - Polling dos módulos de GPS, BME680, YZC-133 (50KG) + HX711 e MPU6050 em intervalos regulares.
    - Leitura de valores de sensores (Ex.: latitude/longitude, níveis de CO_2, contador de ocupação).

4. **Processamento Local**:
    - Detecta superlotamento ou condições perigosas (Ex.: Alertas de lotação, Detecção de frenagem brusca)
5. **Gateway de comunicação**:
    - Empacota dados de sensores em um pacote de dados LoRa e transmite isso para o gateway receptor/servidor.
    - Transmite dados processados de alertas de segurança para o receptor LoRa.

---

5. Protocolo de comunicação

O sistema utiliza principalmente o protocolo LoRa (Long Range) para comunicação sem fio entre o Raspberry Pi Pico e um servidor gateway. O protocolo opera a 900MHz com uma largura de banda de canal de 125kHz, proporcionando conectividade confiável de longo alcance.

Isso permite que o sistema transmita dados de forma eficiente em distâncias de até dezenas de quilômetros, garantindo baixa latência e uso mínimo de largura de banda.

Além disso, o sistema também utiliza o protocolo MQTT (Message Queuing Telemetry Transport) para comunicação de dados em tempo real entre o servidor e o aplicativo móvel. O MQTT é um protocolo leve de mensagens que opera sobre TCP/IP, ideal para dispositivos com recursos limitados e redes de alta latência.

Isso garante que os dados coletados pelos sensores sejam transmitidos de forma eficiente e segura para o servidor, permitindo atualizações em tempo real no aplicativo dos usuários.

Código

O código do projeto Smart Track encontra-se no simulador wokwi conforme link abaixo:
Wokwi Simulator - SmartTrack: Mobilidade Sustentável no Campus

O Código também está anexado em um arquivo compactado zip junto a este relatorio.

Sistema de arquivos

Arquivos do projeto e suas funções:

Arquivo	Função
main.cpp	Logica e loop principal
diagram.json	Wokwi conexões e entidades
bme680.chip.c	Simulador do chip BME680
bme680.chip.json	Config e pinout do BME680
lora-sx1276.chip.c	Simulador do SX1276
lora-sx1276.chip.json	Config e pinout do SX1276

Arquivo	Função
gps-fake.chip.c	Simulador do GPS
gps-fake.chip.json	Config e pinout do GPS
HX711.cpp	Biblioteca adaptada pra Pi Pico SDK
HX711.h	Biblioteca adaptada pra Pi Pico SDK
TinyGPS.cpp	Biblioteca adaptada pra Pi Pico SDK
TinyGPS.h	Biblioteca adaptada pra Pi Pico SDK
utils.h	Macros, funções em comum no codigo
gps.h	Funções de setup e polling do GPS
load_cell.h	Funções de setup e polling do HX711
lora_mqtt.h	Funções de setup e envio de payload MQTT via LoRa
mpu6050.h	Funções de setup e polling da MPU6050

Adaptações de código

Os arquivos acima alguns são 100% de minha autoria e outros são adaptados de outras SDK como arduino para funcionar com a Raspberry Pi Pico W. Os que estão possuem a descrição "Biblioteca adaptada" foram portadas por mim do Arduino para a SDK da Raspberry Pi Pico. Além do custom chip gps-fake que foi usado de um projeto publico no Wokwi [4] e também foi adaptado a biblioteca TinyGPS++ para funcionar com a SDK do Pi Pico ao inves de usar as bibliotecas do Arduino.

Ficou um total de 17 arquivos, sendo 11 de autoria propria e 6 bibliotecas adaptadas. ***Atenção: Em nenhum momento foi utilizado Arduino nesse projeto***.

Limitaçãoes

Os controladores de LoRa, GPS e BME680, Assim como está descrito na tabela de arquivos, todos foram simulados já que não existem por padrão no simulador Wokwi. Isso levou a algumas limitações de simulação devido a complexidade de emular estes chips no Wokwi usando sua API em C [5], além de ser muito demorado e não valer a pena ja que não é o objetivo do projeto a emulação perfeita destes chips.

O simulador Wokwi aparentemente possui limitações de quantidade de linhas de codigo e quando chega neste ponto o simulador para de salvar as alterações. Ou seja, existe uma limitação para a quantidade de coisas a se fazer no simulador que não permite a adição de certos recursos.

Devido a esta limitação de linhas de codigo não foi possível usar o freeRTOS para uma devida gerencia de prioridades de tarefas, assim causando lentidão e desincronizações.

Fora outras alternativas que estavam inicialmente planejado, mas por limitações da Raspberry Pi Pico W não foi possivel fazer (Ex.: Falta do promiscuous mode na Pi Pico SDK) que não seria problema em outras placas.

Infelizmente não houve tempo habil para emulação correta do chip BME680 para captura da temperatura, humidade, emissão de dados de gases e pressão. Sendo assim ficando a desejar na simulação.

Da mesma maneira não outras partes do sistema não poderam ser melhor emuladas como o LoRa SX1276 que foi necessario usar a conexão wi-fi para emular uma conexão de tranmissão e mais rodeios na simulação estatica do uso da biblioteca MQTT.

Mesmo se houvesse tempo habil para a implementação de todas as simulações de controladores e sensores o Wokwi não permitiria salvar o projeto na nuvem pela suposta limitações de quantidades de linha de codigo que podem ser salvas por projeto, testei somente com o plano gratis do Wokwi.

Resultados e Discussão

Resultados

Após a implementação e simulação do código no Wokwi, os seguintes resultados foram observados:

- **Captura de Dados em Tempo Real**: Os modulos de GPS, ocupação (YZC-133 (50KG) + HX711) e movimento (MPU6050) capturaram dados em tempo real com precisão.
- **Transmissão Eficiente**: Os dados foram transmitidos com sucesso via simulação LoRa e MQTT. Que em um cenario real demonstraria baixa latência.
- **Previsões de Ocupação**: O sistema conseguiu processar uma boa estimativa de alunos nos testes efetuados.
- **Interface do Usuário**: A interface de usuario não foi implementada por não estar dentro do escopo pre-estabelecido do projeto final.
- **Receptor/Servidor LoRa**: O Receptor/Servidor LoRa não foi implementada por não ser essencial para a demonstração do funcionamento central do projeto embarcado dadas as condições.
- **Eficiência Energética**: O uso de sensores de baixo consumo e a transmissão LoRa prometem uma operação energeticamente eficiente e adequada.

Deste modo, fica evidente que o SmartTrack pode melhorar significativamente a mobilidade sustentável no campus, proporcionando uma experiência de transporte mais eficiente e confortável para os estudantes.

Discussão

- **Desvantagem**:
    - Não funciona com um ônibus com mais de uma entrada.
    - O sistema embarcado tem que ficar bem perto da entrada do ônibus para que o sinal de peso na entrada não fique inconsistente.
    - Não possui grande fidelidade de quantidade de alunos dentro do onibûs.
    - Os tremores e vibrações do ônibus podem levar a leituras erradas.
    - Todos os sistemas tem que estarem muito bem blindados contra interferências eletromagneticas e climaticas de sol e chuva.

O sistema carece de alguns recursos e redundâncias que fazem falta e geram impresições claras em uma analise mais minusciosa. Tanto no simulador como se estivesse implementado na vida real.

Conclusão

O sistema SmartTrack demonstrou viabilidade teorica e técnica. Embora as limitações dos sensores em condições climáticas adversas exijam desenvolvimento adicional, os recursos de roteamento e transparência

em tempo real representam um avanço significativo nas soluções de mobilidade sustentável no campus.

Referências

1 - TransLoc Duke University
2 - Moovit
3 - IoT Bus Tracking System Localization via GPS-RFID
4 - Wokwi Simulator - GPS Simulator Project
5 - Introdução à API Wokwi Custom Chips C